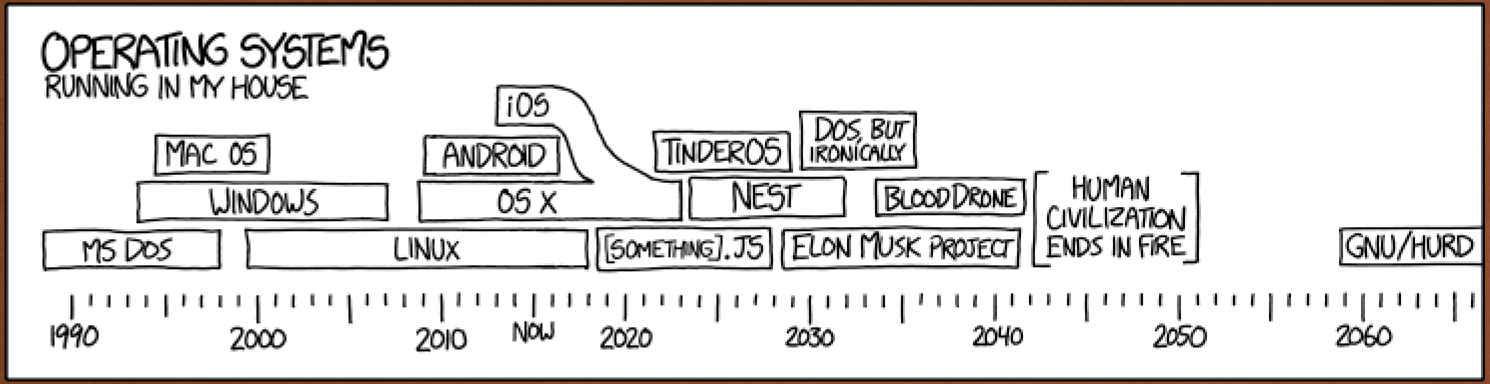
Unit 2 : Operating Systems
Part 1



https://xkcd.com/1508

Unit 2 : Overview

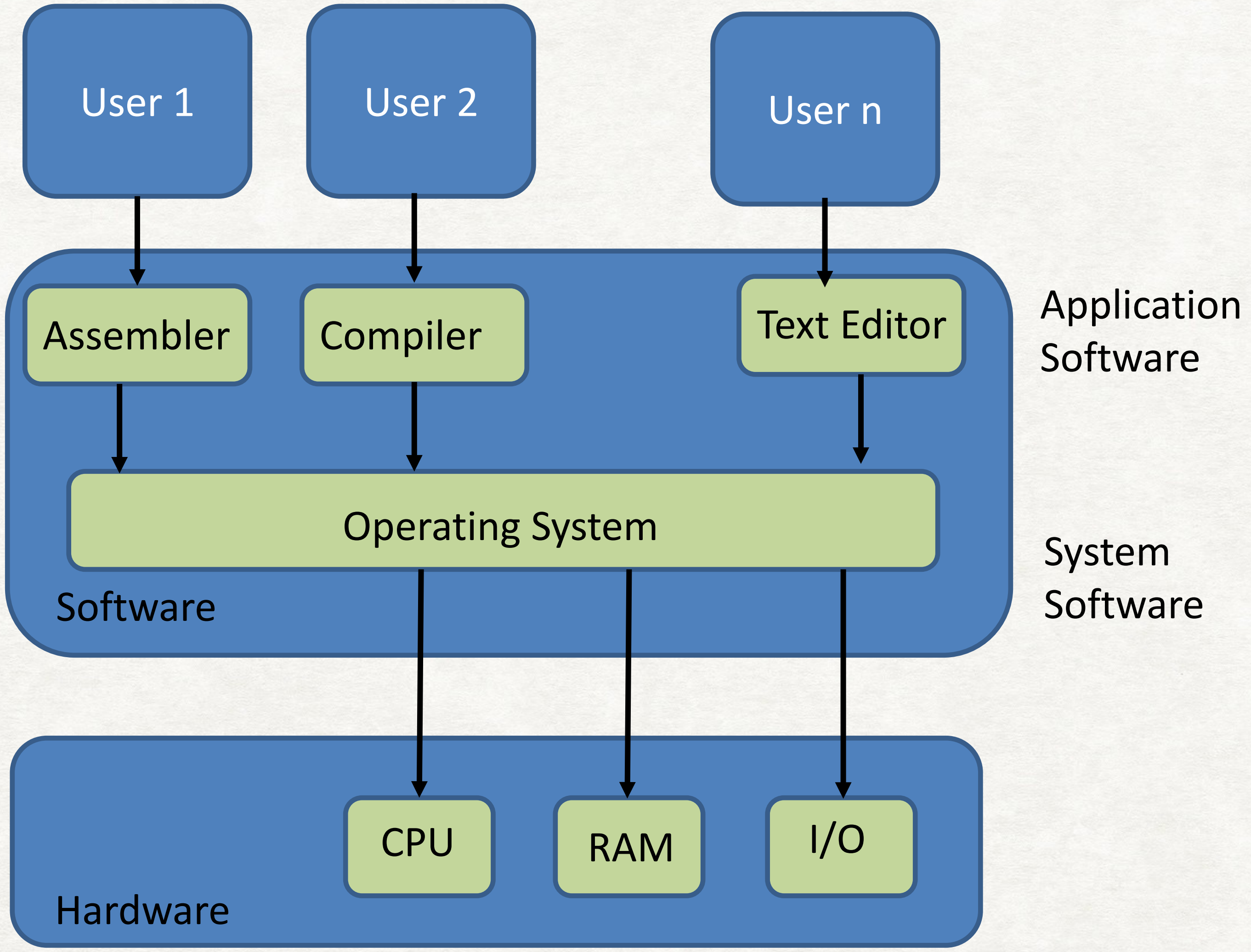
Why do we need an operating system and what does it do?

Different generations of operating systems.

Roles of operating systems.

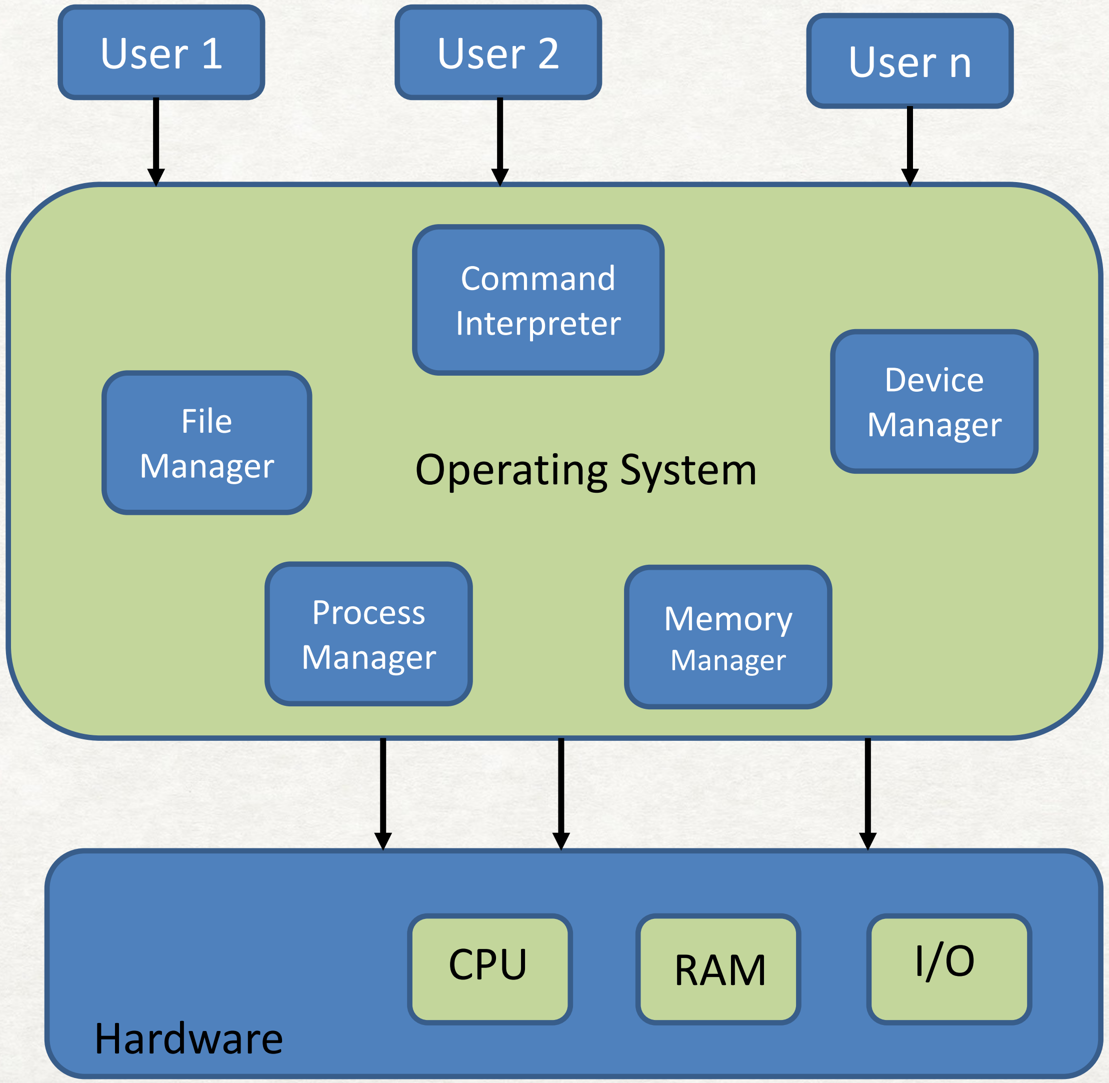
Future of operating systems.

Why do we need an Operating System?



An **Operating System** is a program that manages the computer hardware

What does an OS do?



First generation of operating systems (1945 - 1955)

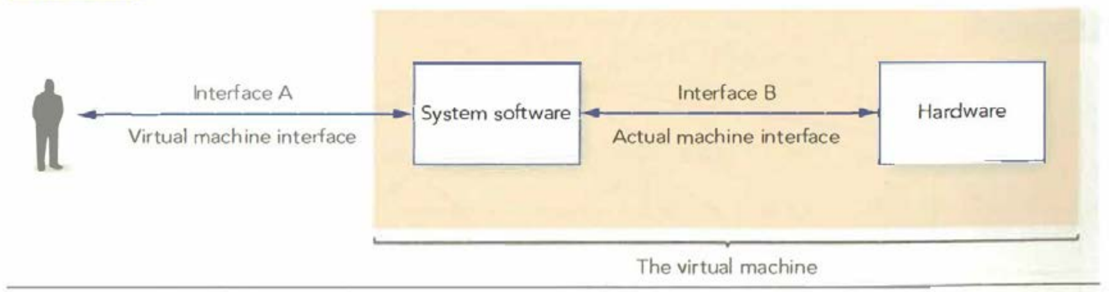
Naked machine

1. Write a program in binary.
2. Load instructions one by one into memory.
3. Insert start into memory address 0 and push “go” button.
4. Read results from memory one by one, in binary.

Virtual machine (A machine with an OS on it).

1. Write a program using a text editor in a high-level language.
2. Save the program to a folder.
3. Use a translator to convert to binary.
4. Use a scheduler to load and run.
5. Use the I/O system to print results.

FIGURE 6.1



The role of system software

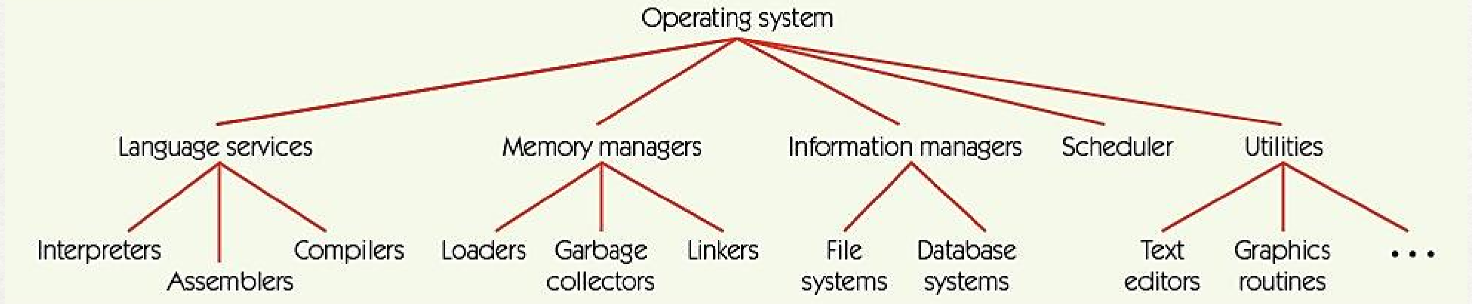
Roles of an OS

1. **User Interface Management (a receptionist)**
2. Control of access to the system and data (a security guard)
3. Program Scheduling and Activation (a dispatcher)
4. Efficient Resource Allocation (an efficiency expert)
5. Deadlock Detection and Error Detection (a traffic officer)

Operating System as a receptionist

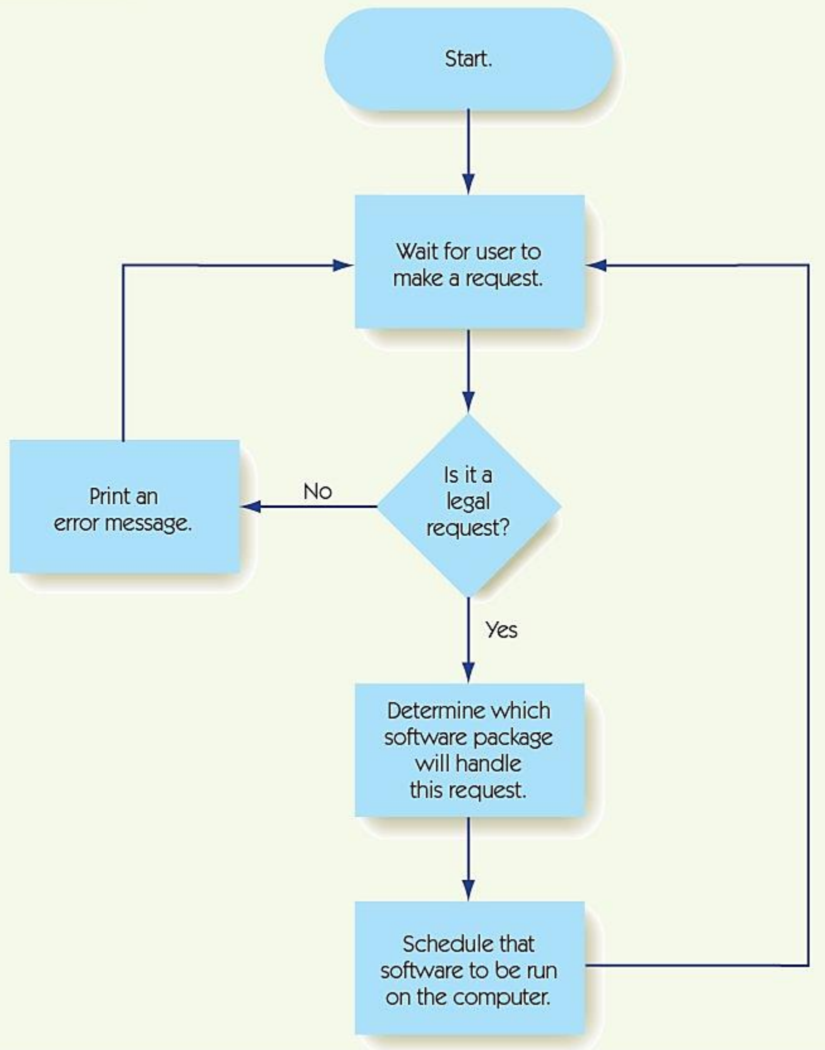
Communicates with users, determines what they want, and activates other system programs, applications, packages, or user programs

FIGURE 6.2



Types of system software

FIGURE 6.15



Start.
Wait for user to make a request.
Is it a legal request?
No
Print an error message.
Yes
Determine which software package will handle this request.
Schedule that software to be run on the computer.

User interface responsibility of the operating system

Interacting with the Operating System

To invoke the services of the Operating System, a user issues ***system commands*** via:

- A command line interface
  - The user types system commands at a prompt in a terminal.
    - We will see examples of some of these commands for both Windows and Linux OS.
  - Command language must be learnt.

- A Graphical user interface (GUI)
  - A visual interface to the virtual machine
  - Makes use of icons/buttons displayed on a screen and selected with a mouse, finger tap, or voice.

Simple System Commands on Windows and Linux

System Command (or Shell Command)	Linux	Windows
Change directory	`cd`	`cd`
Change to parent directory	`cd ..`	`cd ..`
Show present working directory	`pwd`	`echo %cd%`

List directory content
list details
	`ls`
	`ls –al`
		`dir`
		`dir /a`

Create a new directory	`mkdir`	`md`
Remove (delete) directory	`rmdir`	`rd`

Create an empty file	`touch`
	`(Modifies the timestamp of the file)`
		`type nul >` *`filename.txt`*

Create a file and/or edit the content of an existing file	`nano`	`copy con`
		`notepad`

View file content	`less & more`	`type`
Rename and / or move a file	`mv`	`move`
Remove a file	`rm`	`del`
Copy a file	`cp`	`copy`
Show command documentation	`man`	`help`

Roles of an OS

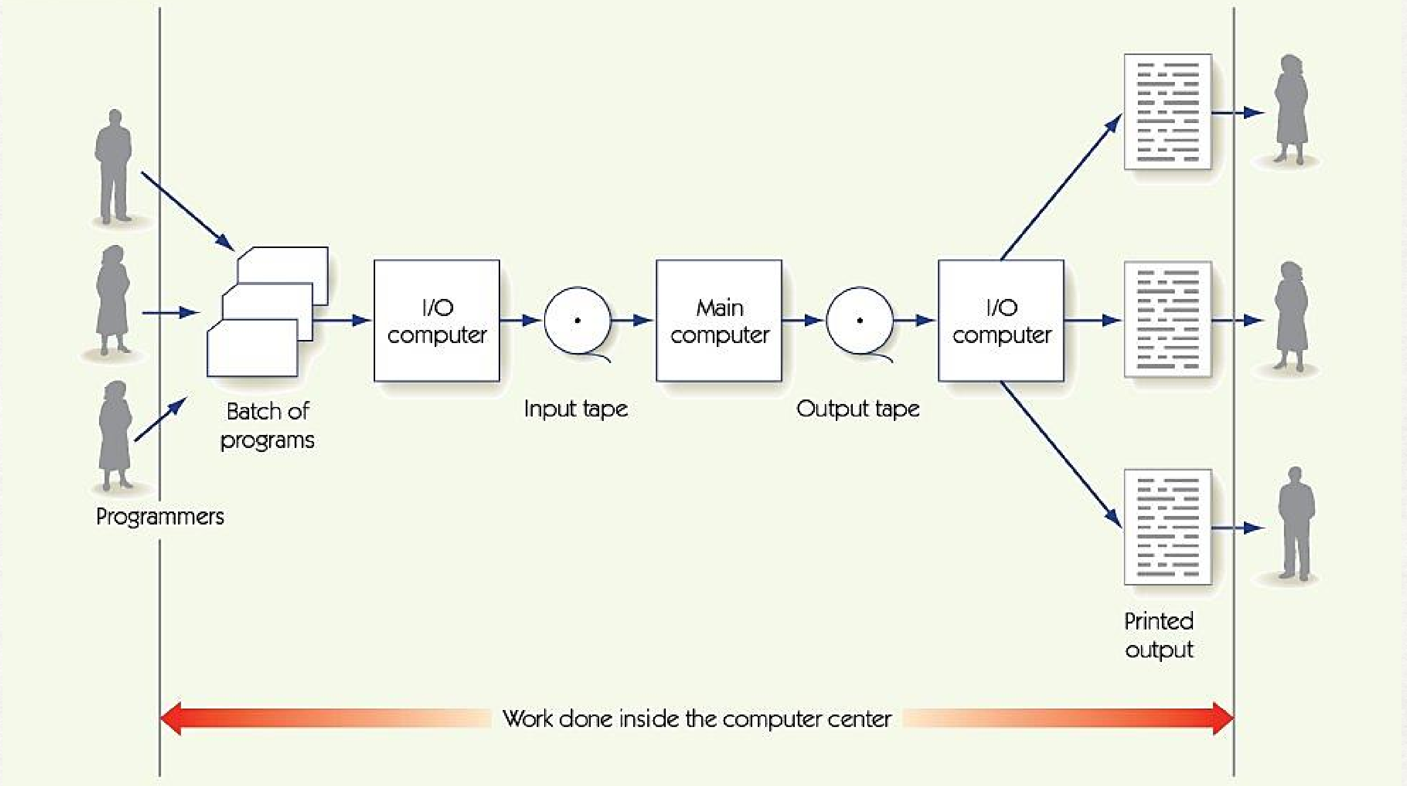
1. User Interface Management (a receptionist)
2. **Program Scheduling and Activation (a dispatcher)**
3. Efficient Resource Allocation (an efficiency expert)
4. Deadlock Detection and Error Detection (a traffic officer)
5. Control of access to the system and data (a security guard)

Second generation: Batch Operating Systems (1955 – 1965)

Early operating systems only knew a **single user,** the operator

- sat at the keyboard (or before that at the card or tape reader),
- grouped programs from many programmers into a "batch"
- loaded these programs onto the actual computer
- returned the output as a printout.

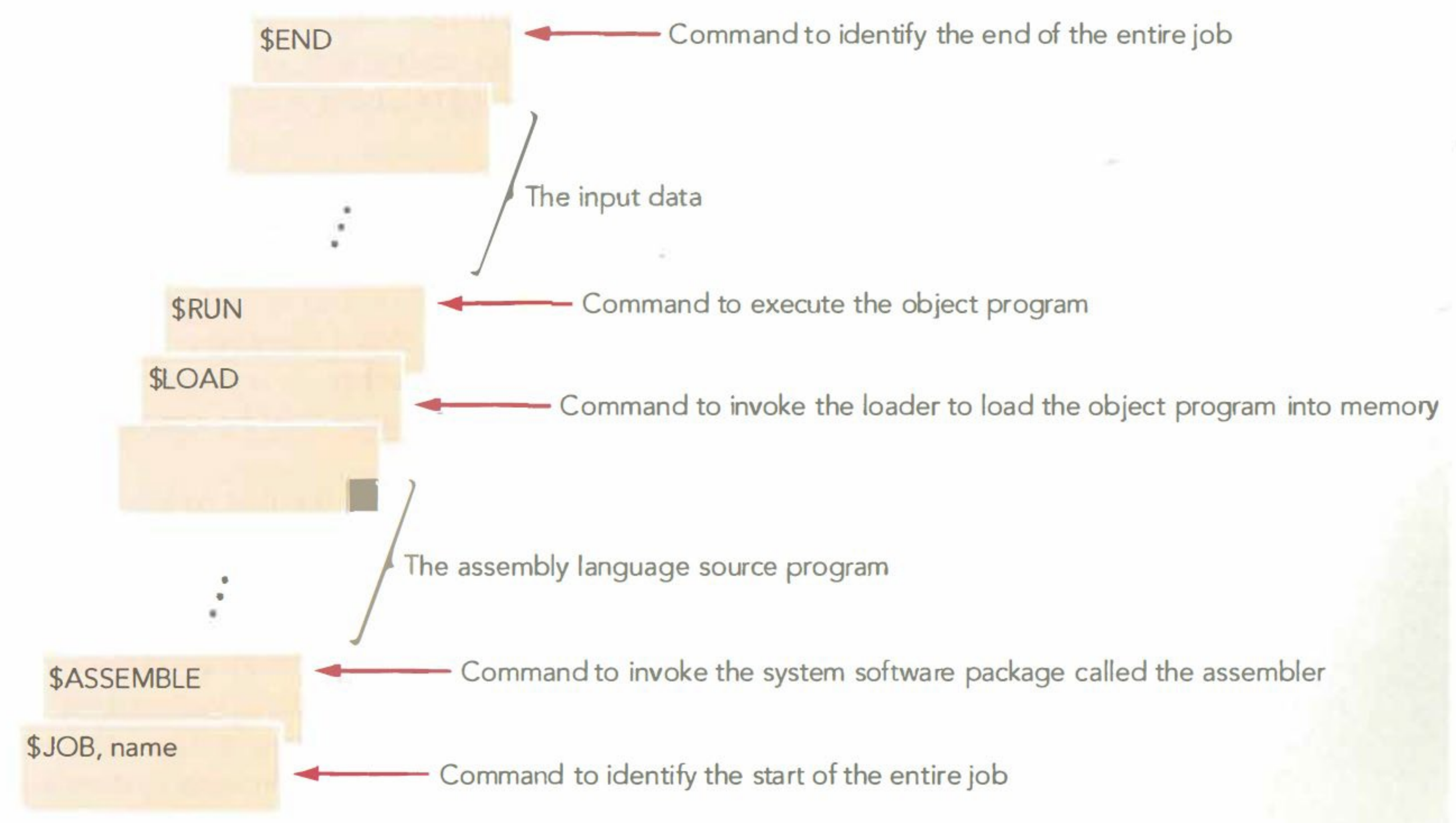
FIGURE 6.17



Operation of a batch computer system

SECOND GENERATION: BATCH OPERATING SYSTEMS (1955 – 1965)

FIGURE 6.18


$END
Command to identify the end of the entire job
The input data
$RUN
Command to execute the object program
$LOAD
Command to invoke the loader to load the object program into memory
The assembly language source program
$ASSEMBLE
Command to invoke the system software package called the assembler
$JOB, name
Command to identify the start of the entire job

Structure of a typical batch job

Let’s do a small exercise.

Assume that in a batch operating system, the loaded program spends on average 25% of its time waiting for I/O.

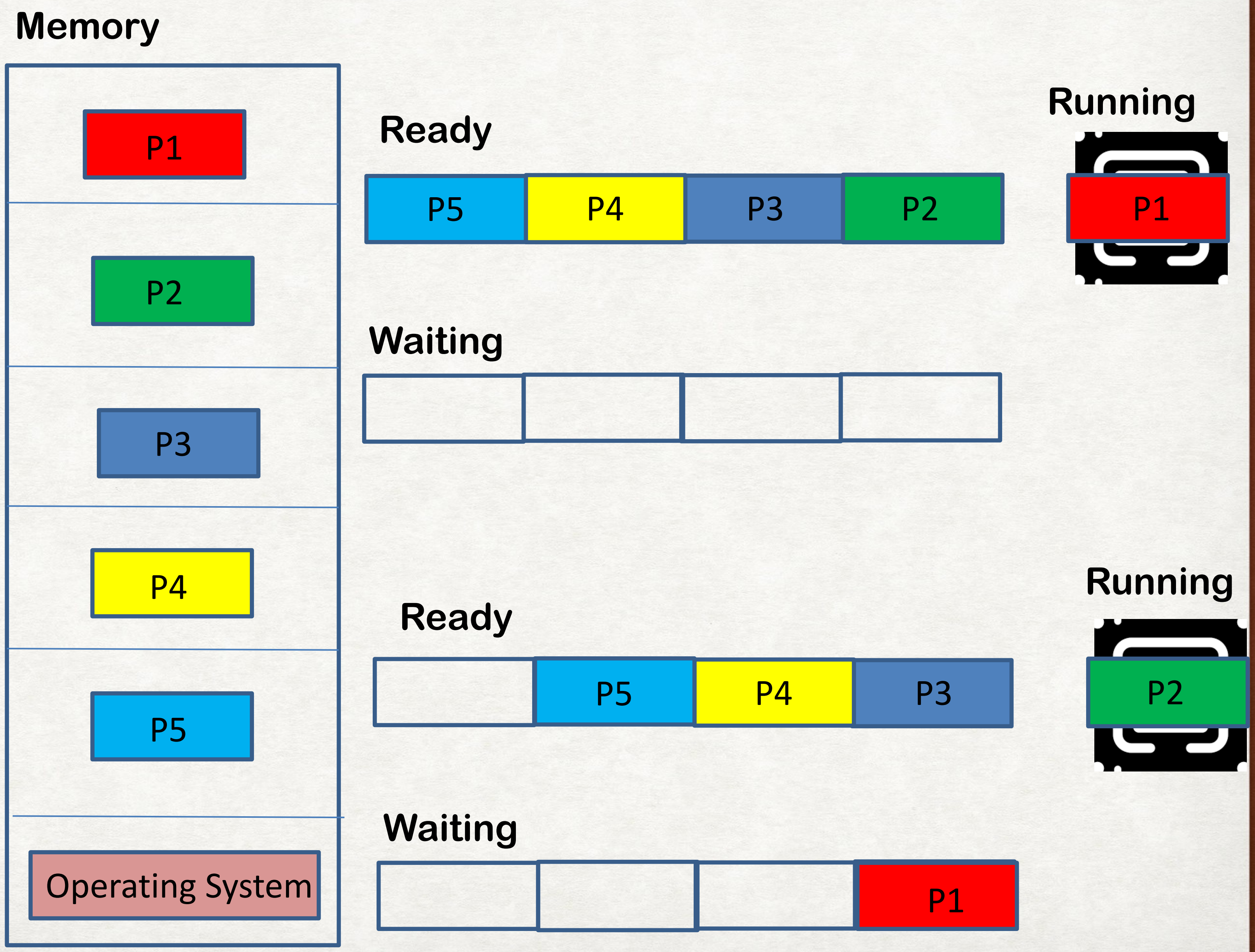
What is the processor utilization?
(Note: **Processor utilization** is the percentage of time the processor is busy.)

The problem with batch OSs

- The batch operating system only kept one program at a time in memory.
  - If that program paused (e.g. for I/O, which could take a few milliseconds) the processor simply waited.
- These idle milliseconds became really significant as computers became faster.
  - giving rise to the next generation of OSs, **multiprogrammed OS**

Third generation : Multi Programmed OS (1965-1985)

- Many user programs (also called ***processes*** or ***jobs***) are loaded simultaneously into the memory.
- **Ensures efficient utilization of CPU time.**
- The additional role of the OS
  - **A Dispatcher!**



Continue with the Exercise!

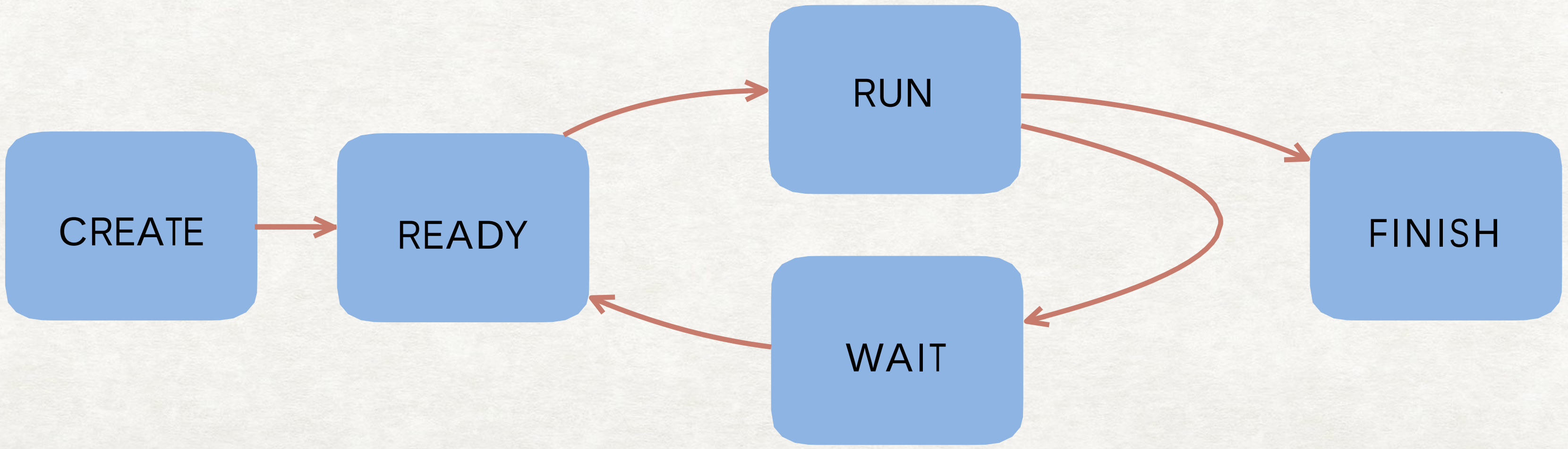
Assume that in a multiprogrammed OS, a program spends 25% of its time waiting for I/O.

If there are 2 programs loaded in memory, what will be the processor utilization?

How much will the processor utilization improve if we have 4 programs instead of 2?

OPERATING SYSTEM AS A DISPATCHER / SCHEDULER

- It is the job of the **scheduler (or process manager)** to choose which process should run next.
- Processes are "ready to run" when they have all the resources, they need in order to run.
- When a running process waits or finishes the scheduler chooses the process which will run next.

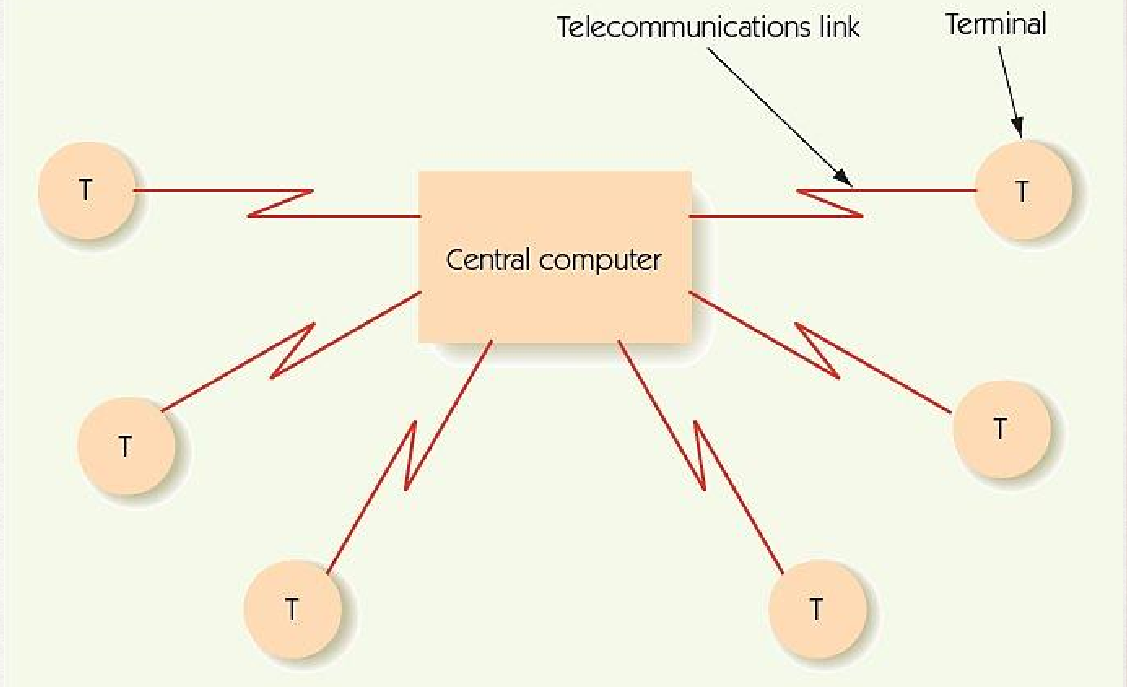


Process State Diagram

Time- Shared OS (1970s)

- Multiprogrammed OS, but users are on the system interactively.
- Users need an illusion of sole access.
  - OS sets up **“virtual machines”** on which other programs run.
- Allocate run time in ***time slices*** - each program runs until it needs an I/O OR its time runs out.

FIGURE 6.19



Configuration of a time-shared computing system